Oussama ELKACHOINDI
Wajdi MEHENNI



CREATION D'UN SITE D'ENCHERES OPENCCM ET EJB

Rapport du Projet de Fin d'Etudes — ASR 2005/2006

Encadrant : Mme Sophie Chabridon

SOMMAIRE

Introduction

Dans le cadre du projet de fin d'études option ASR, nous devons créer deux versions simplifiées d'un site de gestion d'enchères l'un utilisant les composants EJB et l'autre les composants CORBA (CCM) et de comparer les avantages et inconvénients de ces 2 modèles de composants.

Notre application permet à des utilisateurs de s'enregistrer et de poser par la suite des produits en enchère ou de participer à des enchères. L'utilisateur ne peut accéder aux fonctionnalités du site qu'après avoir été identifié (en entrant son pseudo et son adresse email). A l'issue de son identification, une session lui est attribuée et est fermée lorsqu'il se déconnecte. Le site offre à chaque utilisateur la possibilité de connaître toutes les enchères faites sur un produit spécifique et d'avoir un historique concernant les enchères qu'il a déjà faites. On s'est contenté, dans le cadre de ce site, de fournir une interface graphique simple.

Dans ce rapport, nous allons exposer les démarches suivies dans les deux versions pour développer l'application et nous allons présenter surtout les phases d'installation des produits utilisés et les manières de les exploiter. Nous allons aussi exposer des jeux de tests que nous avons effectués pour mesurer les performances de cette application.

I- Utilisation des composants EJB

1. Choix technologique

1.1. Une architecture à trois couches

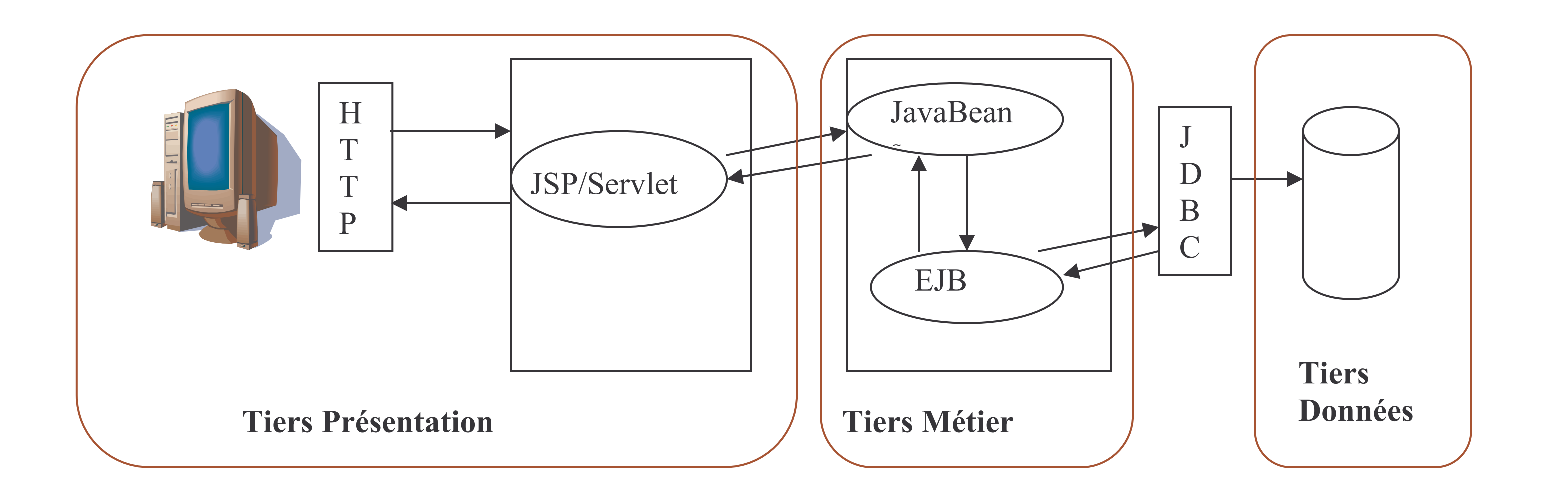
Nous avons essayé de suivre le modèle de développement J2EE. Nous avons trois couches :

-**Couche présentation** : La couche de présentation assure la logique de navigation mais aussi la gestion des droits de l'utilisateur (authentification, session).

-**Couche métier** : Implémentée sous forme de JavaBeans ou EJB, c'est dans cette couche que l'on retrouve l'ensemble des traitements d'une application.

-**Couche persistance** : Elle se compose souvent d'une base de données de type SGBDR.

Ceci permet d'avoir un code plus simple et une implémentation plus claire où il y a une séparation entre la présentation des données et leur traitement. Ainsi des pages JSP se chargent de l'affichage et du formatage des données alors que des Java Beans et des EJB se chargent d'effectuer les calculs.



1.2. Le serveur d'application et la base de données

1.2.1 Environnement de travail

Pour mettre en place notre application, il nous a fallu les pré-requis suivants :

-SDK J2SE que l'on récupère sur le site officiel de Sun. Nous avons utilisé la version 1.4.2_07.

-Ant, un Make multi-plateforme basé sur Java que l'on récupère sur le site officiel d'Apache. Nous avons utilisé la version 1.6.2.

-BCEL, une librairie nécessaire à Ant pour compiler que l'on récupère sur le site officiel du projet Jakarta d'Apache. Nous avons utilisé la version 5.1

-Jonas que l'on récupère sur le site officiel d' ObjectWeb. Nous avons utilisé la version 4.1.

-Tomcat que l'on récupère sur le site d'Apache. Nous avons utilisé la version 5.0.25.

1.2.2 Installations

Pour mettre en place l'environnement de travail, nous avons suivi la procédure suivante :

-Créer un répertoire Appli/ sous votre $HOME.

-Dézipper dans ce répertoire les fichiers `apache-ant-1.6.2-bin.tar.bz2`, `jakarta-tomcat-5.0.25.tar.gz`, `jonas.tgz` et `bcel-5.1.tar.gz.`

–Installer, dans ce répertoire, la jdk 1.4.

-Créer un répertoire appServer sous le répertoire $HOME.

-Mettre à jour le .bashrc de la façon suivante :

```
export AppliHOME=$HOME/Appli
```

```
export JONAS_ROOT=$AppliHOME/JONAS_4_1
```

```
export JONAS_BASE=$HOME/appServer

export CATALINA_HOME=$AppliHOME/jakarta-tomcat-
5.0.25

export CATALINA_BASE=$JONAS_BASE

export PATH=$JONAS_ROOT/bin/unix:$PATH

export JAVA_HOME=$AppliHOME/j2sdk1.4.2_07

export PATH=$JAVA_HOME/bin:$PATH

export ANT_HOME=$APPLI_HOME/apache-ant-1.6.2

export PATH=$ANT_HOME/bin:$PATH
```

-Ouvrir un nouveau terminal ou taper « bash » pour que le système tienne compte du nouveau .bashrc.

- Copier le fichier bcel-5.1.jar dans le répertoire $ANT_HOME/lib

-Aller dans le répertoire $JONAS_ROOT/ et taper « ant create_jonasbase »

-Vérifier que l'environnement de travail est correct en tapant « jonas check ».

-Pour démarrer le serveur Jonas, taper « jonas start ». Si tout se passe bien, on aurait l'affichage suivant :

```
JONAS_BASE set to /mci/ei0306/elkachoi/appServer
```

```
17:40:21,151 : PolicyProvider.init : Using JOnAS PolicyConfigurationFactory provider and JOnAS Policy provider

17:40:21,463 : TraceCarol.infoCarol : Name service for jrmp is started on port 1099

17:40:21,510 : ServiceManager.startRegistry : registry service started

17:40:21,921 : MBeanServerFactory.createMBeanServerImpl : Created MBeanServer with ID: 54a328:1090cbc9e18:-7ffd:elaphe.int-evry.fr:1

17:40:22,174 : RMIConnectorServer.start : RMIConnectorServer started at: service:jmx:rmi://localhost/jndi/jrmpconnector_jonas

17:40:22,176 : ServiceManager.startJmx : jmx service started

17:40:23,068 : ServiceManager.startServices : jtm service started

17:40:23,198 : HsqlDBServiceImpl.doStart : Starting HSQLDB server 1.7.2 on port 9001

17:40:24,007 : HsqlDBServiceImpl.doStart : HSQLDB server started.

17:40:24,053 : HsqlDBServiceImpl.doStart : User not found: SA

17:40:24,054 : HsqlDBServiceImpl.doStart : Dropping and adding user 'jonas' with password 'jonas'.

17:40:24,057 : HsqlDBServiceImpl.doStart : Error while creating/adding user : 'null'.

17:40:24,059 : ServiceManager.startServices : db service started

17:40:24,137 : DataBaseServiceImpl.createDataSource : Mapping ConnectionManager jdbc:mysql://www-tp.int-evry.fr/jonas_db1 on jdbc_1

17:40:24,158 : ServiceManager.startServices : dbm service started

17:40:24,823 : ServiceManager.startServices : security service started

17:40:25,763 : JmsAdminForJoram.startMOM : starting MOM on host localhost, port 16010

17:40:25,765 : JmsAdminForJoram.start : starting JmsAdmin with host localhost, port 16010

17:40:26,270 : ServiceManager.startServices : jms service started

17:40:26,890 : Rar.processRar : /mci/ei0306/elkachoi/appServer/rars/autoload/JOnAS_jdbcCP.rar available

17:40:27,044 : Rar.processRar : /mci/ei0306/elkachoi/appServer/rars/autoload/JOnAS_jdbcXA.rar available

17:40:27,117 : Rar.processRar : /mci/ei0306/elkachoi/appServer/rars/autoload/JOnAS_jdbcDM.rar available

17:40:27,227 : Rar.processRar : /mci/ei0306/elkachoi/appServer/rars/autoload/JOnAS_jdbcDS.rar available

17:40:27,232 : ServiceManager.startServices : resource service started

17:40:28,463 : JContainer.addBean : MEJB available

17:40:28,486 : ServiceManager.startServices : ejb service started
```

```
17:40:29,695 : Http11Protocol.init : Initialisation de Coyote HTTP/1.1 sur http-9000

17:40:29,777 : StandardService.start : DÃ©marrage du service Tomcat-JOnAS

17:40:29,789 : StandardEngine.start : Starting Servlet Engine: Apache Tomcat/5.0.25

17:40:29,865 : StandardHost.start : XML validation disabled

17:40:29,994 : Http11Protocol.start : DÃ©marrage de Coyote HTTP/1.1 sur http-9000

17:40:32,570 : AbsJWebContainerServiceImpl.registerWar : War
/mci/ei0306/elkachoi/appServer/webapps/autoload/ctxroot.war available at the context /.

17:40:34,439 : AbsJWebContainerServiceImpl.registerWar : War
/mci/ei0306/elkachoi/appServer/webapps/autoload/jonasAdmin.war available at the context /jonasAdmin.

17:40:34,450 : ServiceManager.startServices : web service started

17:40:34,499 : ServiceManager.startServices : ear service started

Le serveur JOnAS 'jonas' version 4.1 est actif

17:40:34,502 : Server.start : Le serveur JOnAS 'jonas' est dÃ©marrÃ© sur  rmi/jrmp
```

- Pour faire des tests, nous avons téléchargé le répertoire des exemples du cours à l'adresse http://www-inf.int-evry.fr/COURS/CORBA_gb/Site/ , et l'avons mis sous le répertoire $JONAS_BASE.

Remarque importante : Pour des applications où on a besoin d'avoir une connexion à une base de données, il faut penser à télécharger le driver correspondant.

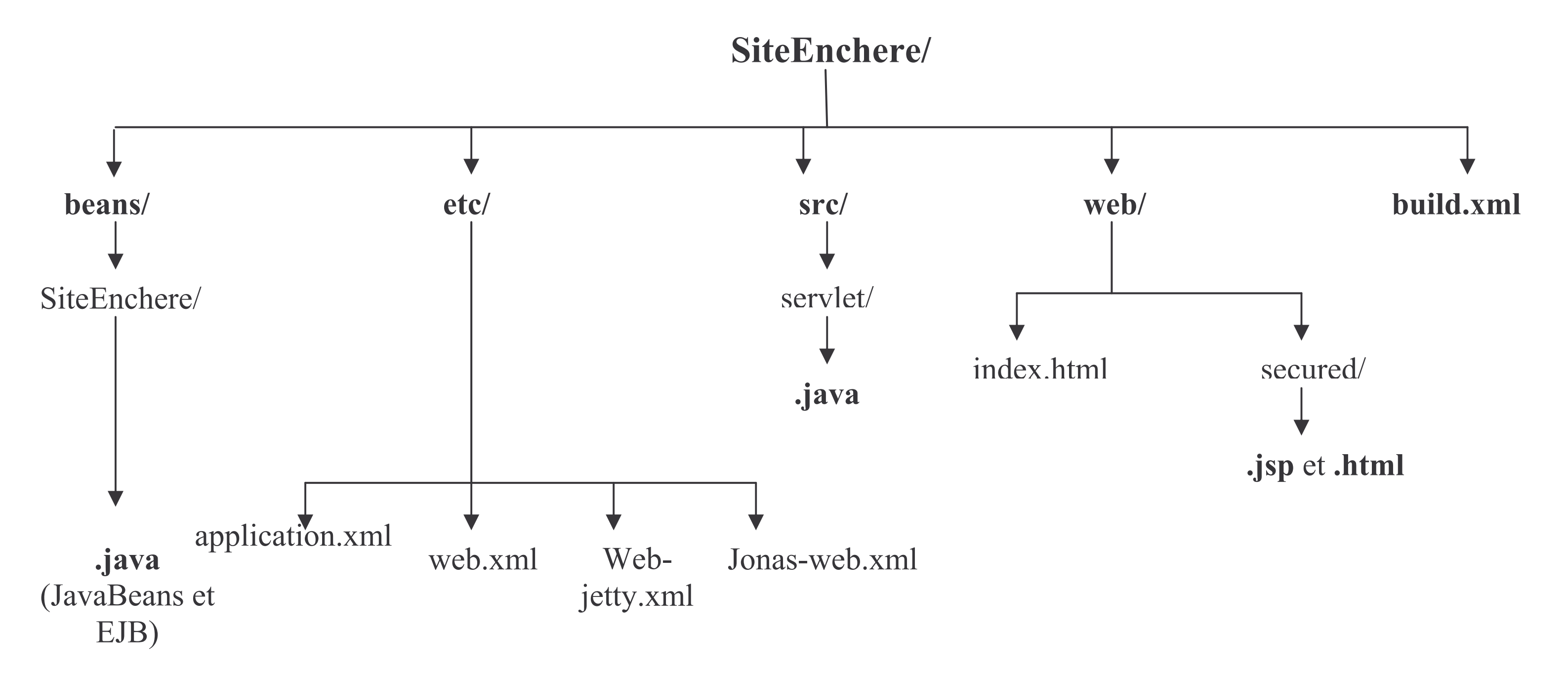
Par exemple, dans notre cas, nous avons eu besoin d'une base de données MySQL et pour cela, il a fallu télécharger le driver ***mysql-connector.*** Plus précisément, il faut juste copier le ficher ***mysql-connector-java-3.0.14-production-bin.jar*** dans le répertoire **$JONAS_ROOT/lib/ext**/.

1.3 Déploiement de l'application Web sur le serveur Jonas

1.3.1. Une arborescence bien définie

Vu que dans notre application nous avons travaillé sur les trois couches de l'architecture J2EE, c'est à dire nous avons utilisé des fichiers JSP, des JavaBeans et des EJB avec un accès à une base de données, le déploiement est un peu spécial.

Il faut avoir une architecture bien définie des répertoires et des fichiers. Nous exposons ici l'arborescence des répertoires de notre application et nous l'expliquons par la suite :



Le répertoire ***beans/ :*** Ce répertoire doit contenir les fichiers de la couche métier c'est-à-dire les JavaBeans et les EJB. Dans notre cas, nous avons réuni ces fichiers dans un seul package SiteEnchere.

Le répertoire ***etc/ :*** Ce répertoire contient des fichiers de configuration :

- **application.xml** : permet et de spécifier les fichiers .jar at .war.
- **web.xml :** permet de donner une description de l'application.
- **web-jetty.xml :** permet de configurer le Realm pour Jetty.
- **Jonas-web.xml :** spécifie les ressources Jonas.

Le répertoire ***src/ :*** Ce répertoire contient le code source des servlets. Dans notre cas, nous avons une seule servlet « ControleurAcces.java » sous le package servlet.

Le répertoire ***web/ :*** Ce répertoire contient d'une part le fichier « index.html » (la page par défaut du site) et d'autre part, un répertoire que nous avons nommé **secured/.** Celui-ci contient les fichiers JSP et HTML de l'application.

Le fichier ***build.xml :*** Ce fichier est obligatoire pour compiler l'application avec l'outil Ant.

1.3.2. Compilation

La compilation de l'application s'effectue en utilisant l'outil Ant. Il suffit d'aller dans le répertoire SiteEnchere/ et de taper ant.

1.3.3. Exécution

Pour exécuter l'application :

-Lancer le serveur jonas : jonas start.

-Ouvrir un navigateur Web et taper http://<hote>:<port>/jonasAdmin :

<hote> est le nom de la machine serveur.

<port> c'est le port sur lequel est configuré le serveur Jonas, généralement 9000.

Dans notre cas, http://elaphe.int-evry.fr:9000/jonasAdmin

-Entrer «jonas » comme pseudo et mot de passe.

-A gauche, cliquer sur le lien ***Applications (EAR).***

-Sélectionner « SiteEnchere.ear » puis cliquer sur le bouton >>>

-Cliquer sur ***Apply*** puis sur ***Confirm***.

On vient ainsi de déployer notre application. S'il y a des erreurs dans les fichiers XML qui se trouvent dans le répertoire **etc/,** une erreur apparaît lors de l'appui sur le bouton « Confirm ».

-Pour visiter le site, il ne reste plus alors qu'à entrer l'adresse URL correspondante : http://<hote>:<port>/<nom_application> où <nom_application> est le nom qui précède l'extension .ear. Dans notre cas, http://elaphe.int-evry.fr:9000/SiteEnchere
On tombe sur la page « index.html ».

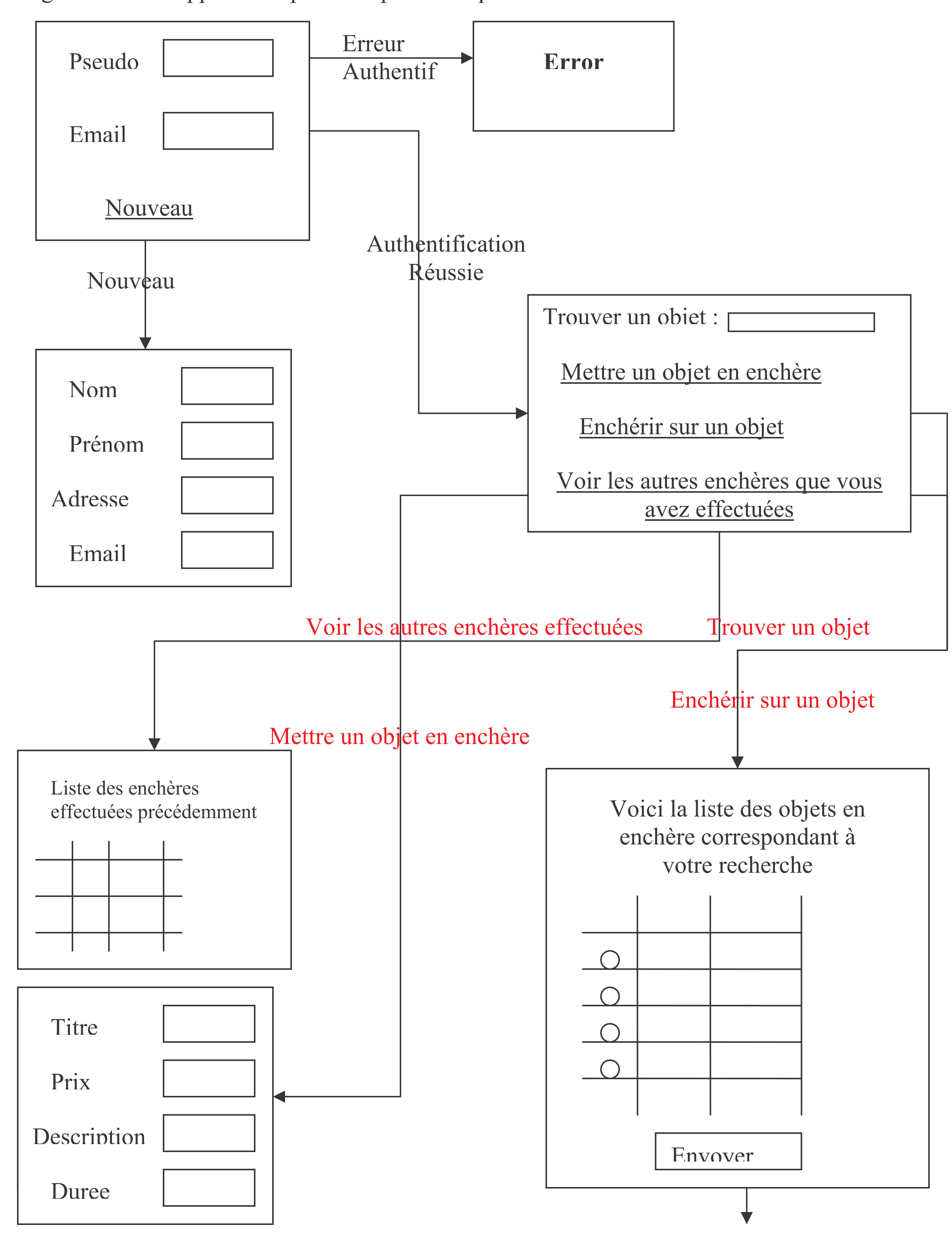
1.4. La Base de données

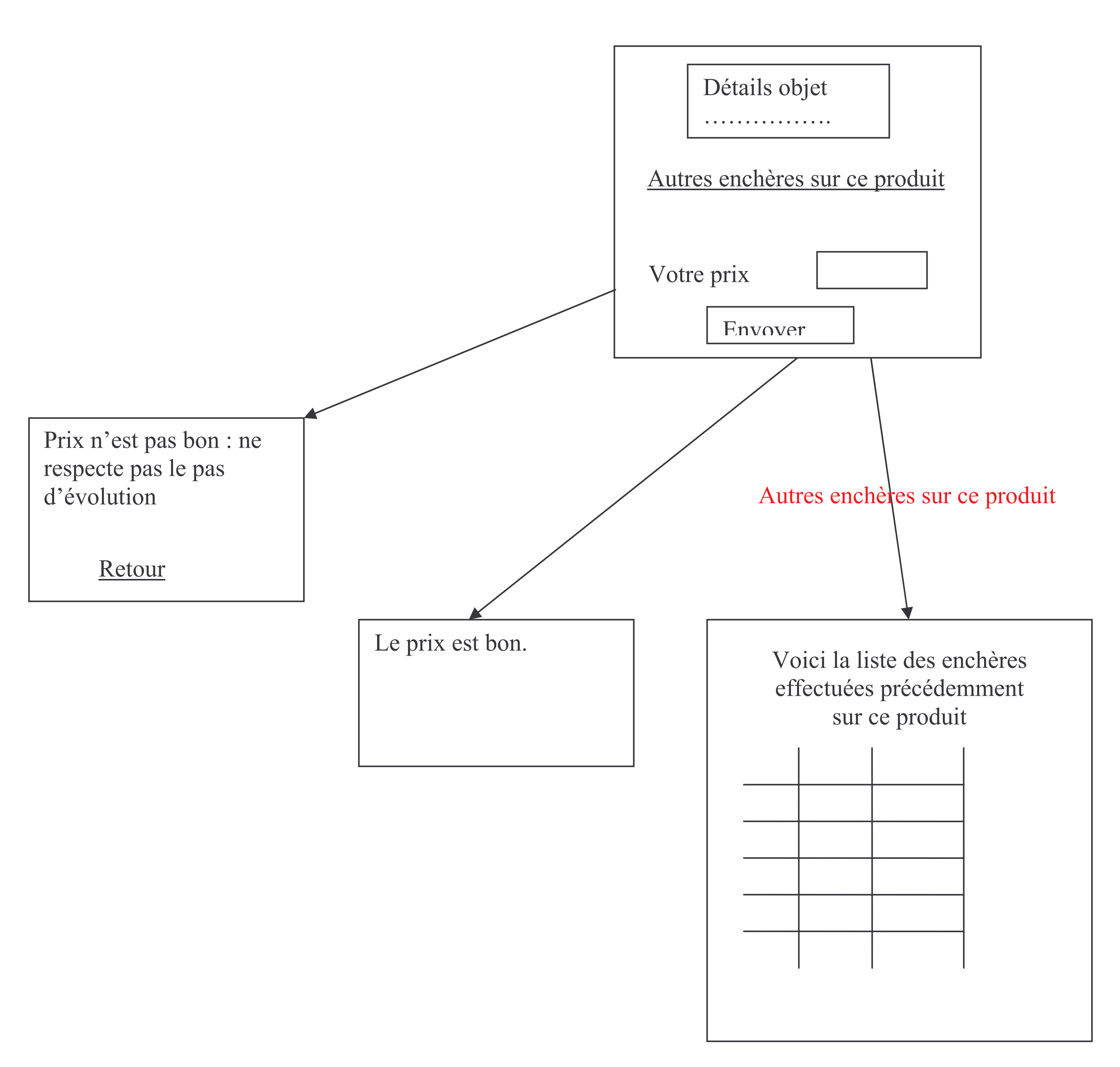
Nous avons utilisé une base de données MySQL. C'est celle qui se trouve sur la machine « www-tp » de l'INT. Pour y accéder taper http://www-tp.int-evry.fr/phpMyAdmin/ puis « jonas_db1 » comme pseudo et « ujdb1 » comme mot de passe.

2. Conception de l'interface Utilisateur

Nous nous sommes contentés d'une interface simple mais fonctionnelle. Son développement est basé sur des balises HTML et JSP. Une touche créative pourra être ajoutée par la suite grâce à des images d'arrière-plan, pour les boutons, des cadres, etc.

L'ergonomie de l'application peut être présentée par le schéma ci-dessous :





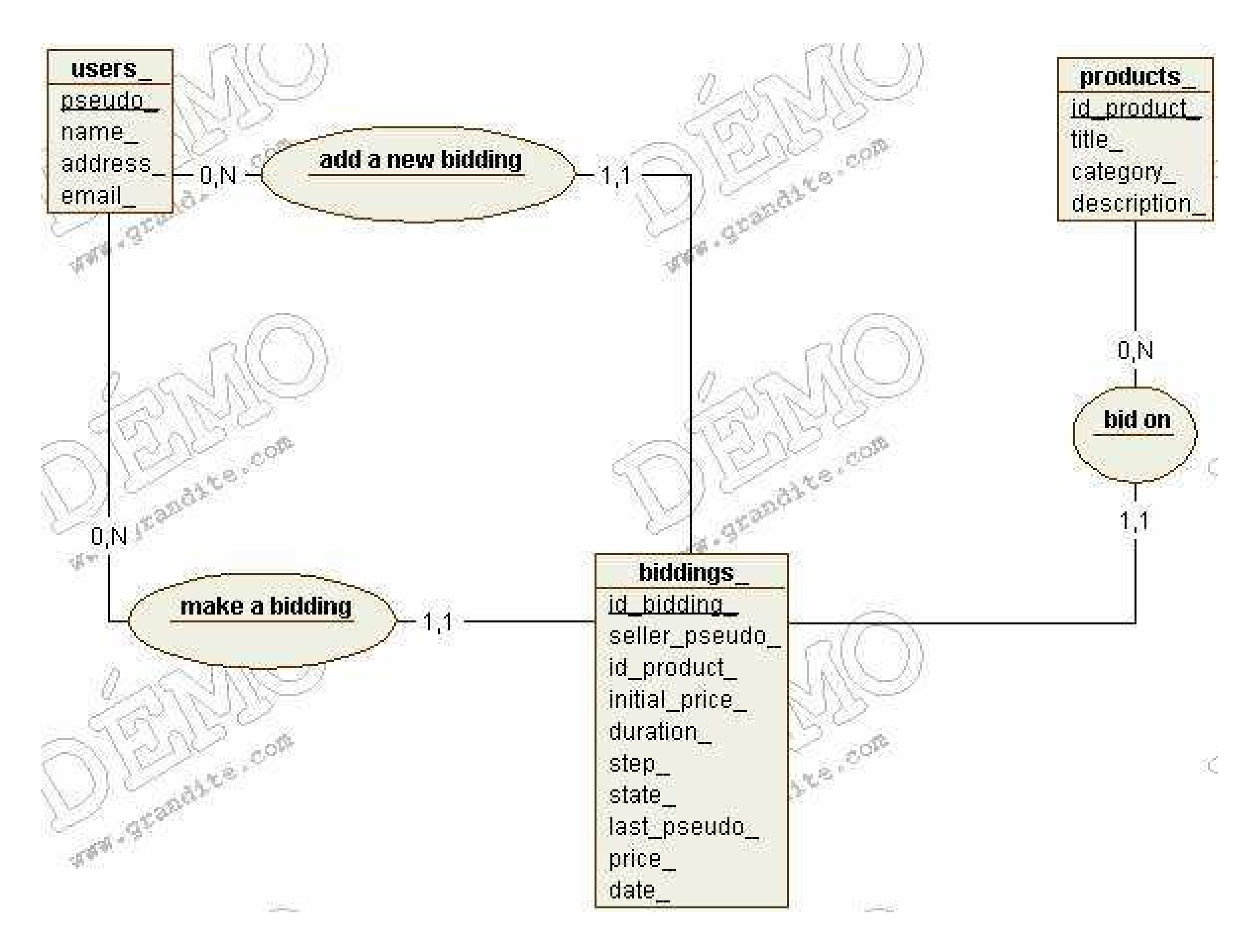
L'utilisateur commence par entrer son pseudo et son email (s'il en possède sinon il doit s'inscrire). On vérifie si les informations sont bonnes et si c'est le cas il sera dirigé vers un écran qui lui permettra de choisir entre la mise d'un objet en enchère, la consultation des objets qui sont mis en enchère ou la consultation de l'historique des enchères qu'il a déjà effectuées. Selon son choix, il sera dirigé vers un écran lui demandant de saisir les informations concernant l'objet qu'il veut mettre en enchère, un écran lui listant les produits disponibles en enchère ou un écran lui listant les enchères qu'il a effectuées précédemment.

Dans le deuxième cas, l'utilisateur peut choisir un objet sur lequel il peut enchérir. Il choisira dans un second écran le prix avec lequel il pose son enchère sachant que ce prix doit respecter le pas d'évolution choisi par le vendeur. L'utilisateur peut aussi consulter l'historique des enchères posées sur le produit qu'il a choisi.

Dans tous ces écrans les informations saisies par l'internaute sont contrôlées et dans le cas d'un problème il est redirigé vers une page d'erreur.

3. Conception de l’application

3.1 Conception de la base de données



Notre base de données est composée de trois tables : **users_** qui modélise un utilisateur en tant que vendeur ou acheteur, **products_** qui modélise un produit mis en enchère et **biddings_** qui modélise une enchère. Une enchère peut être interprétée de deux façons : du point de vue d’un vendeur, il s’agit de la mise en enchère d’un produit et du point de vue d’un acheteur où il s’agit de poser une enchère sur un produit.

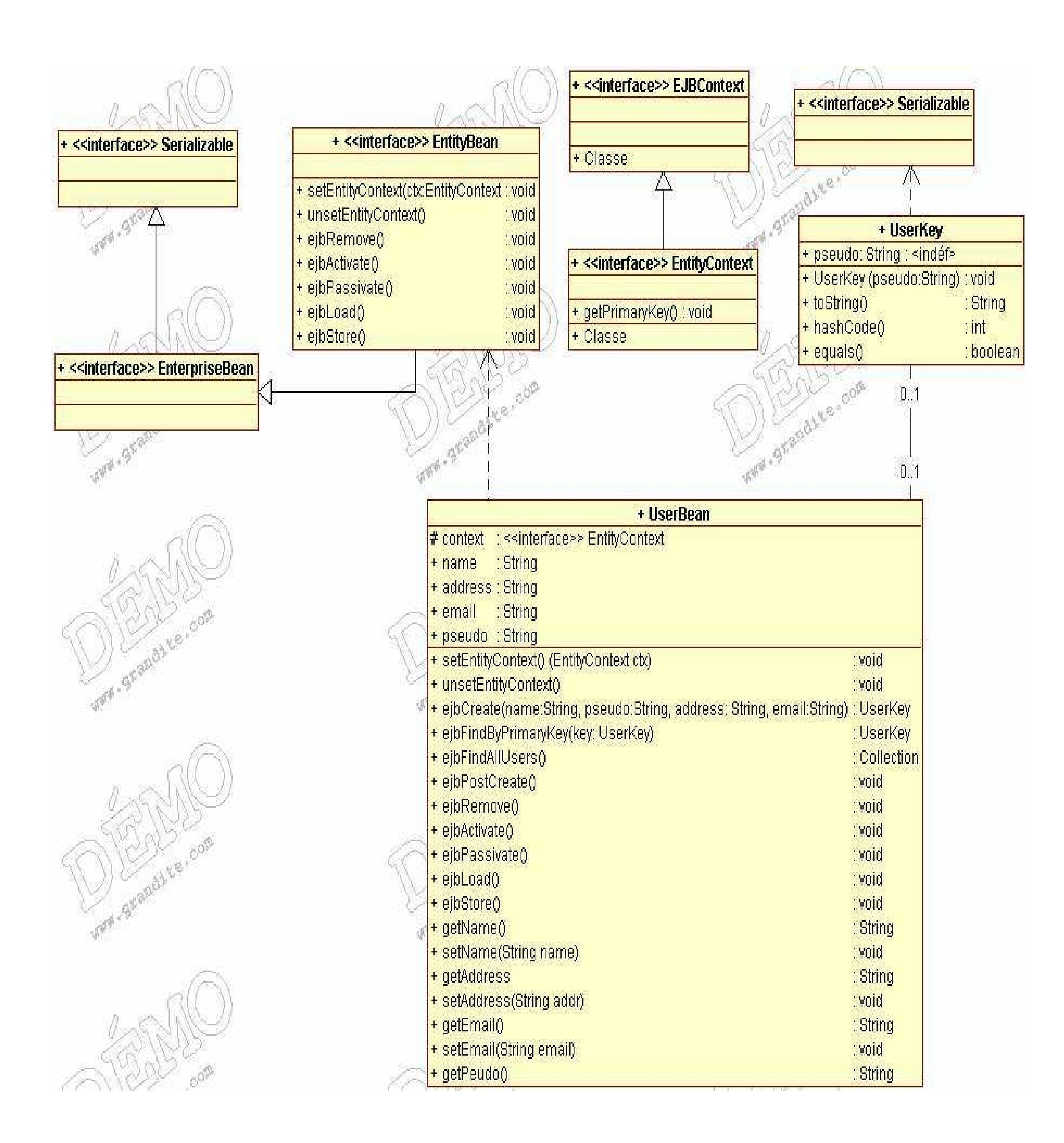
Un utilisateur est identifié par son pseudo, un produit par un id qui est incrémenté automatiquement et une enchère par un id incrémentée automatiquement aussi. D’après ce diagramme, on remarque bien qu’un utilisateur en tant que vendeur peut poser de 0 à N enchères (c'est-à-dire qu’il peut mettre en enchère 0 à N produits), et en tant d’acheteur peut aussi participer à 0 ou N enchères (c'est-à-dire qu’il peut enchérir sur 0 à N objets). A un instant donné, une enchère ne peut correspondre qu’à un seul utilisateur et un seul produit. En outre, un produit peut faire l‘objet de 0 à N enchères.

3.2 Conception des EJB

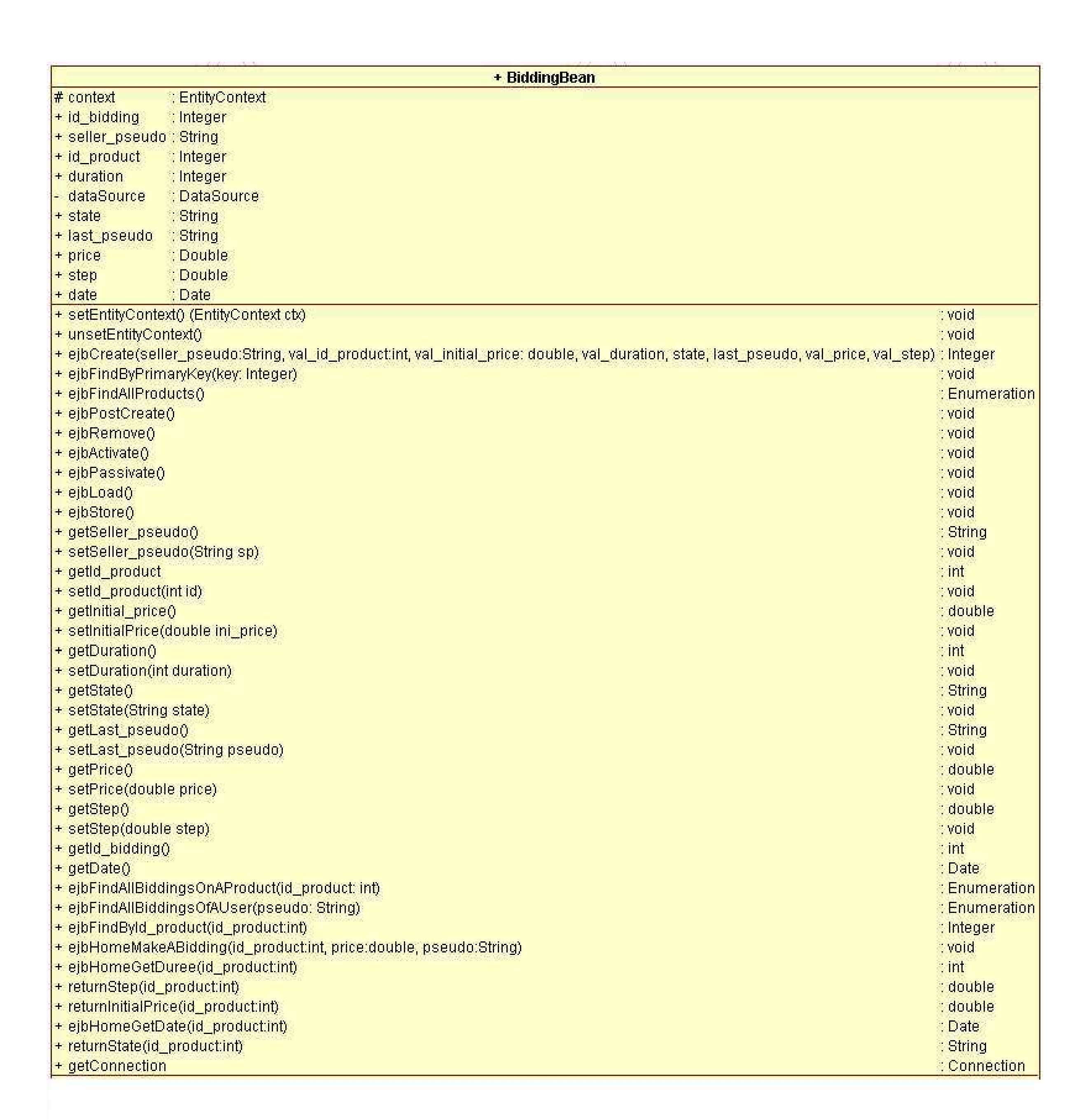
Nous avons déjà conclu que la couche métier de notre application est formée par des EJB et des JavaBeans. Les objets utilisés sont des objets persistants : beans d'entité.

Nous avons eu besoin de trois beans d'entité : le bean **UserBean** qui correspond à la table Users_ de notre base de données, le bean **BiddingBean** qui correspond à la table biddings_ et le bean **ProductBean** qui correspond à la table products_.

Nous donnons ici le digramme de classe complet correspondant au bean UserBean et juste la classe principale pour les beans BiddingBean et ProductBean car leurs diagrammes de classes sont semblables à celui de UserBean.



+ ProductBean	
# context	: EntityContext
+ id_product	: Integer
+ title	: String
+ category	: String
+ description	: String
+ dataSource	: DataSource
+ setEntityContext(EntityContext ctx)	: void
+ unsetEntityContext()	: void
+ ejbCreate(title:String, category:String, description:String)	: ProductKey
+ ejbFindByPrimaryKey(key:ProductKey)	: ProductKey
+ ejbFindAllProduct()	: Enumeration
+ ejbFindProductsByCriteria(criterium:String)	: Enumeration
+ ejbPostCreate()	: void
+ ejbRemove()	: void
+ ejbActivate()	: void
+ ejbPassivate()	: void
+ ejbLoad()	: void
+ ejbStore()	: void
+ getTitle()	: String
+ setTitle(title:String)	: void
+ getCategory()	: String
+ setCategory(cat:String)	: void
+ getDescription()	: String
+ setDescription(description:String)	: void
+ getId_product()	: int
+ setId_product(Intger id)	: void
+ getConnection()	: Connection



Comme on peut le remarquer à travers ces diagrammes, nous avons utilisé une persistance BMP (Bean Managed Persistence) c'est-à-dire que nous avons géré nous-mêmes les interactions avec la base de données.

Le bean « UserBean » possède quatre attributs (name, address, email et pseudo). En plus des méthodes de récupération et de définition de ces attributs (les getters/setters), il possède la méthode **ejbFindByPrimaryKey(key : UserKey)** qui permet de retourner une instance utilisateur selon sa clé primaire (le pseudo) et la méthode **ejbFindAllUsers()** qui permet de retourner tous les utilisateurs.

Le bean « ProductBean » a exactement la même architecture.

Le bean « BiddingBean » est plus compliqué car il constitue le lien entre les deux autres et le centre des fonctionnalités de notre site. En plus des méthodes classiques concernant la création, la récupération, la suppression d'un EJB et des méthodes de récupération et définition des attributs, ce bean possède la méthode **ejbFindAllBiddingsOnAProduct(id_product : int)** qui permet de retourner toutes les enchères faites sur un produit spécifique, la méthode **ejbFindAllBiddingsOfAUser(pseudo : String)** qui permet de retourner toutes les enchères qu'a faites un utilisateur, **ejbFindById_product(id_product : int)** qui permet de retourner la dernière enchère faite sur un produit, **ejbHomeMakeABidding(id_product :int, price :double, pseudo :String)** qui permet d'insérer une enchère faite par un utilisateur sur un produit avec un prix précis, **ejbHomeGetDuree(id_product :int)** qui permet de calculer restante avant l'expiration de la période d'exposition d'un produit et la méthode **ejbHomeGetDate(id_product :int)** qui retourne la date de la mise en enchère d'un produit. Ce bean fait appel à d'autres méthodes non accessibles par le client mais qui sont indispensable pour la méthode « ejbHomeMakeABidding » : **returnStep(id_product :int)** qui retourne le pas de progression d'un produit, **returnInitialPrice(id_product :int)** qui retourne le prix initial d'un produit et **returnState(id_product :int)** qui retourne l'état d'un produit (encore disponible ou pas).

3.3. Conception des JavaBeans

Les JavaBeans jouent le rôle d'intermédiaire entre les EJB, qui manipulent les données de la base de données, et les JSP qui présentent ces données sous une certaine forme. Nous avons utilisé trois JavaBeans : **Login.java** qui gère l'ajout des utilisateurs dans la base de données, **NewProduct.java** qui gère l'ajout d'un produit et **ProductList.java** qui permet de lister les produits en enchère, les enchères effectuées par un utilisateur ou les enchères effectuées sur un produit spécifique. Ces beans sont utilisés par plusieurs pages JSP. Le but est d'éviter de faire des appels aux EJB dans ces fichiers JSP : tous ces appels se font à l'intérieur des JavaBeans. Nous présentons ci-dessous ces différents beans :

+ Login	
- pseudo	: String
- email	: String
- name	: String
- address	: String
+ Login()	: void
+ addUser()	: User
+ isUsed()	: boolean
+ setPseudo(String pseudo)	: void
+ setEmail(String email)	: void
+ setName(String name)	: void
+ setAddress(String address)	: void

La méthode **addUser()** récupère une instance du HOME de UserBean et ajoute un utilisateur à la base grâce à la méthode « create ». La méthode **isUsed()** récupère aussi une instance du HOME de UserBean et elle vérifie si le pseudo choisi par l'utilisateur est déjà utilisé ou pas, en utilisant la méthode « controlePseudo » du bean.

+ NewProduct	
- pseudo	: String
- title	: String
- category	: String
- description	: String
- initial_price	: double
- duration	: int
- step	: double
+ NewProduct()	: void
+ addProduct()	: void
+ setPseudo(String pseudo)	: void
+ setTitle(String title)	: void
+ setCategory(String category)	: void
+ setDescription(String description)	: void
+ setInitial_price(double initial_price)	: void
+ setDuration(int duration)	: void
+ setStep(double step)	: void

Comme pour « addUser » du JavaBean précédent, la méthode « addProduct » utilise une instance du HOME de ProductBean pour ajouter un produit dans la base de données.

+ ProductList	
- available_products	: Vector
- prices	: Vector
- durations	: Vector
- states	: Vector
- steps	: Vector
- dates	: Vector
- last_biddings_user	: Vector
- user_dates	: Vector
- last_biddings	: Vector
- title	: String
- description	: String
- category	: String
- price	: double
+ ProductList()	: void
+ listProducts()	: void
+ listBiddingsOnAProduct(id_product:int)	: void
+ listBiddingsOfAUser(pseudo:String)	: void
+ productDetails(id_product:int)	: void
+ getProductName(id_product:int)	: String
+ controlPrice(price:double, id_product:int, BiddingHome bhome)	: boolean
+ synchronized insertNewBidding(id_product:int, price:double,pseudo:String)	: String
+ returnStep(id_product:int)	: double
+ sendMail(to:String, content:String)	: void
+ getBuyerMail(pseudo:String, ctx: initialContext)	: String
+ putDateIntoFormat(seconds:int)	: String
+ searchProducts(criterion: String)	: void

NB : Sur ce dernier diagramme, nous n'avons pas présenté les getters/setters des attributs par souci de clarté.

La méthode **listProducts()** permet de lister les produits qui sont mis en enchère. Elle enregistre leurs titres, prix, états, durées et pas de progression dans les vecteurs available_products, prices, states, durations et steps. Ces vecteurs seront par la suite lus par le fichier JSP qui se chargera d'afficher ces produits. **listBiddingsOnAProduct()** et **listBiddingsOfAUser** se basent sur le même principe mais affichent respectivement les enchères qui ont été faites précédemment sur un produit donné et les enchères qui ont été faites par un utilisateur. La méthode **productDetails** se charge de récupérer les détails d'un produit (cette méthode est utilisée au moment où l'utilisateur a choisi un produit sur lequel il va enchérir). La méthode **insertNewBidding()** permet d'enregistrer une nouvelle enchère dans la base de données.

4. Implémentation des fonctionnalités principales de l'application

4.1. Enregistrement d'un utilisateur et l'ajout d'un produit

Comme nous l'avons déjà mentionné, l'accès à notre site se fait après l'introduction d'un pseudo et d'une adresse email. Si l'utilisateur ne possède pas encore de compte il peut créer un en cliquant sur le lien **« Si vous êtes nouveau cliquer ici. »** de la page d'accueil. Ceci le mène vers un formulaire HTML où il peut saisir ses informations personnelles. Les données saisies seront traitées dans le fichier « addNewUser.jsp ». Celui-ci utilise le bean « Login.java » pour enregistrer le nouvel utilisateur :

```
<jsp:useBean id="login" scope="session" class="SiteEnchere.Login" />

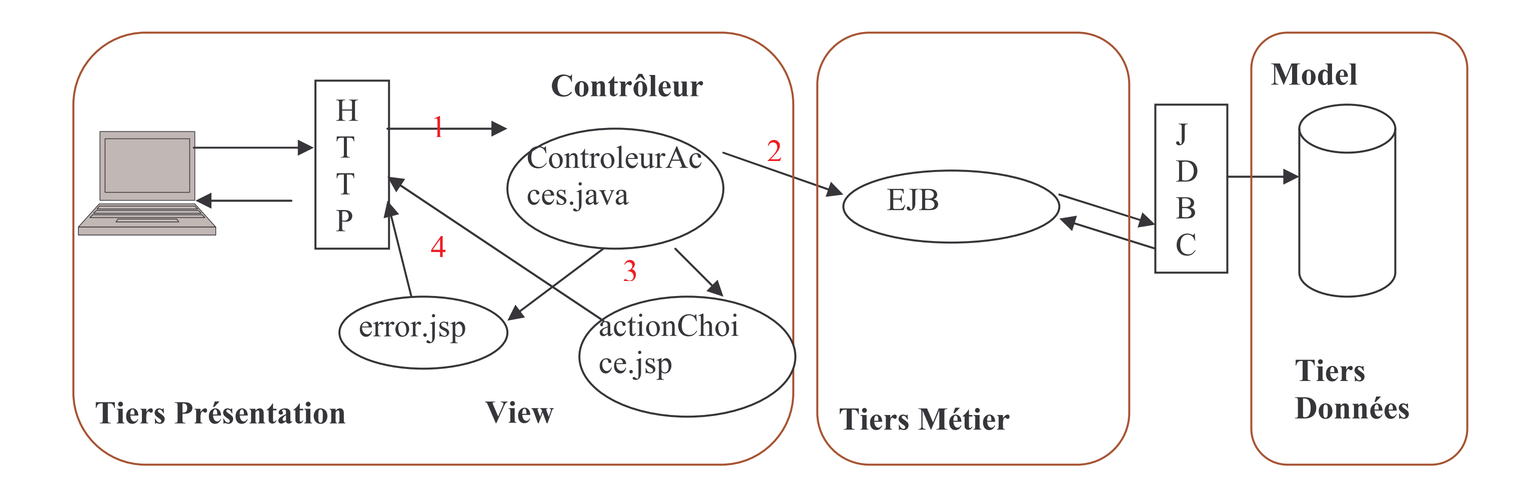
      <!--Verify the parameters-->
      <%
      if(!(request.getParameter("name")).equals("") &&
      request.getParameter("name")!=null &&
      !(request.getParameter("address")).eq\
      uals("") && !(request.getParameter("pseudo")).equals("") &&
      !(request.getParameter("email")).equals("")){
      %>
      <jsp:setProperty name="login" property="*"/>
      <%
              User user=login.addUser();
              if(user!=null){//The pseudo and email are valuable
                      session.setAttribute("user",user);
      %>
```

Une fois que qu'on l'a enregistré, on crée une session qui lui correspond. Cette session sera fermée au moment où il se déconnecte.

En ce qui concerne l'ajout d'un nouvel produit, nous avons suivi le même principe : un formulaire HTML pour la saisie des informations concernant ce produit et un fichier JSP (« addNewProduct.jsp ») qui utilise le bean « NewProduct.java » pour l'ajout du produit dans la base de données en cas de succès.

4.2. L'authentification

L'authentification se fait par l'intermédiaire d'un formulaire HTML au niveau de la page d'accueil. L'utilisateur doit saisir son pseudo et son mot de passe. Ces données saisies seront redirigées vers la servlet « ControleurAcces.java » pour les traiter. En effet, nous avons essayé ici de suivre le modèle MVC (Model, View, Controller) où la base de données joue le rôle de Model, la servlet le rôle de Controller et les pages JSP le rôle de View :



1 : L'utilisateur émet une requête auprès du serveur en utilisant la méthode POST.
2 : Le contrôleur accède aux informations demandées au travers d'un JavaBean.
3 : Le contrôleur transmet le JavaBean à la page JSP chargée de la mise en forme du document.
4 : La page JSP utilise les informations du JavaBean transmit, met en forme les informations et retourne la réponse à l'utilisateur.

La servlet utilise l'EJB «UserBean » pour vérifier si les données saisies par l'utilisateur sont bonnes ou pas.

4.3. L'affichage des produits en enchère et le contrôle de leur état

Ceci se fait au niveau du fichier «BiddingOnProduct.jsp ». Les produits sont présentés sous forme de tableau. Chaque ligne correspond à un produit et fournit les informations le concernant tel que le titre, le prix actuel, la durée restante avant l'expiration de la période de son exposition… Tant que cette durée est non nulle, un bouton radio est ajouté au début de chaque ligne permettant ainsi à l'utilisateur de choisir le produit correspondant pour enchérir dessus. Ces informations sont récupérées des vecteurs « available_products », «prices », « durations », « states » et « steps » qui sont remplis par le bean « ProductList.java » :

```
<%
for(int i=0;i<available_products.size();i++){
        Product pdt=(Product)available_products.elementAt(i);
        Double price=(Double) prices.elementAt(i);
        double val_price=price.doubleValue();
        Integer duration=(Integer) durations.elementAt(i);
        int val_duration=duration.intValue();
        String state=(String) states.elementAt(i);
        Double step=(Double) steps.elementAt(i);
        double val_step=step.doubleValue();
        if(state.equals("true")){
%>
```

```
	<!--The product is still under bidding-->
	<tr><td><INPUT TYPE="radio" NAME="product" VALUE="<%=
pdt.getId_product()%>"></td><td> <%= pdt.getId_product()%> </t\
d><td><%=pdt.getTitle()%></td><td><%=pdt.getDescription()%></td><td>
<%=val_price%></td><td><%=list.putDateIntoFormat(val_duration)%></td
><td><%= state%></td><td><%= val_step%></td></tr>
<%
	}else{
%>
%>
	<!--The product is not still under bidding-->

<tr><td></td><td><%=pdt.getId_product()%></td><td><%=pdt.getTitle()%
></td><td><%=pdt.getDescription()%></td><td><%=val_price%></td><td><
%=list.putDateIntoFormat(val_duration)%></td><td><%=state%></td><td>
<%= val_step%></td></tr>
```

C'est la méthode "listProducts" du bean "ProductList" qui se charge, entre autre, de contrôler la date d'expiration de la période de présentation du produit en enchère et d'envoyer un mail au dernier enchérisseur. En effet, cette méthode récupère la durée restante d'exposition du produit à l'aide de la méthode « getDuree() » de l'EJB BiddingBean.

```
	Integer duration = new Integer(bhome.getDuree(pdt
							.getId_product()));
```

Ensuite, elle récupère la dernière valeur de l'état du produit enregistrée dans la base de données. Si cette valeur est « false » alors aucun traitement n'est fait (car ceci suppose qu'un mail est déjà envoyé au meilleur enchérisseur), sinon si la durée récupérée précédemment est inférieure à 0 alors elle récupère le pseudo du dernier enchérisseur et son adresse email et lui envoie un mail lui confirmant l'achat du produit. Si la durée d'exposition du produit a expiré et aucune enchère n'a été faite la dessus, un mail est envoyé à son vendeur pour l'informer que son produit n'a pas été vendu. A l'issue de ce traitement la valeur de l'état du produit est mise à « false » dans la base de données :

```
if (b.getState().equals("true")) {
	if (duration.intValue() <= 0) {
		state = "false";
		b.setState("false");
		String byerPseudo = b.getLast_pseudo();
		if (!(byerPseudo.equals(b.getSeller_pseudo()))) {
			String message = "Bonjour," + "\n"+ "Vous avez achete le
			produit "+ pdt.getTitle()
			+ " sur notre site d'enchere.\n Il coÃ»te "
			+ price + "\nAu revoir.";
			String mail = getByerMail(byerPseudo,initialContext);
			sendMail(mail, message);
		} else {// the product was not bought. we send a mail to
// its seller.
			String message = "Bonjour,"
			+ "Nous sommes désolés le produit  " + pdt.getTitle()
			+ " que vous avez pose en enchère sur notre site d'enchère,
a dépassé sa durée d'exposition et n'a pas été vendu.\nAu revoir."
	}
```

4.4. Insertion d’une nouvelle enchère et contrôle du prix choisi par l’utilisateur

L’utilisateur entre le prix avec lequel il veut enchérir au niveau de la page « biddingTreatment.jsp ». Ce prix sera transmis à « bid.jsp » qui fait appel au bean « ProductList » et plus particulièrement à la méthode «insertNewBidding » pour vérifier si le prix entré respecte le pas de progression ou pas et insérer l’enchère dans la base de donner si c’est le cas. Cette méthode vérifie aussi que celui qui fait l’enchère n’est pas le vendeur du produit… Elle retourne une chaîne de caractère qui est égale à «price not available » si le prix choisi ne respecte pas le pas de progression, «buyer not valid » si celui qui fait l’enchère est le vendeur du produit et « price available » sinon. Cette chaîne de caractère est renvoyée au fichier JSP « bid.jsp » qui redirige alors l‘utilisateur vers un écran lui confirmant le succès de l’enchère qu’il a faite ou vers un écran lui demandant d’entrer un autre prix car le prix entré ne respecte pas le pas de progression ou encore vers un écran qui lui informe qu’il ne peut pas enchérir sur le produit s’il est son vendeur :

```
      Success= « price not available » ;
      try {

          Bidding b = bhome.findById_product(id_product);
//verify that the buyer is not the seller_pseudo
          if(pseudo.equals(b.getSeller_pseudo())){
                    success="byer not valid";
          }
          else{
            utx.begin();
            //verify that the chosen price respects the step
            boolean availability = controlPrice(price, id_product,
bhome);
            if (availability == true) {

                         bhome.makeABidding(id_product, price, pseudo);
                         success = "price available";
             }
             utx.commit();
            }
          } catch (Exception e) {
                      System.err.println("Cannot create Bidding: " + e);
                      System.exit(2);
                      }

                      return success;


      }
```

5. Les Tests

5.1. Principe

Il s’agit de tester la réaction du serveur lorsque la charge monte sur certains liens principaux où l’accès à la base de données est important. Pour cela nous avons implémenté un programme de test « Bench.java » qui permet de lancer plusieurs threads envoyant simultanément des requêtes HTTP vers une adresse donnée. Ce programme permet aussi de mesurer le temps total qu’a mis le serveur pour répondre à toutes ces requêtes.

5.2. Le gestionnaire de connexion poolMan

Dans notre application nous avons retenu l’approche la plus simple pour accéder à la base de données consistant à créer une nouvelle connexion pour chaque requête. Ceci peut entraîner un blocage système lorsque de nombreux utilisateurs veulent accéder aux données du site. Nous avons pensé alors à l’outil poolMan qui est un gestionnaire de connexions qui crée et manipule des regroupements de connexions à la base de données. Toutes les connexions d’un même regroupement sont dirigées vers le même URI JDBC (même hôte, même instance de base de données, même id de connexion). Ceci permet alors d’alléger le trafic sur la base de données.

Pour utiliser poolMan il faut :

-Télécharger poolMan sur ce site http://sourceforge.net/projects/poolman/

-de- zipper le fichier.

-ajouter dans le CLASSPATH tous les .jar qui se trouvent dans le répertoire poolman-2.1-b1/lib.

-Mettre le Driver de connexion à la base de données dans le répertoire $JAVA_HOME/jre/lib/ext

-Copier poolman.jar dans $JONAS_BASE/examples/output/webapps/<nom_de_l’application>/WEB-INF/lib

(nom_de_l’application est dans notre cas SiteEnchere)

-Ecrire un fichier poolman.xml comme suit : (les paramètres varient bien sûr d’un cas à un autre)

```
<?xml version="1.0" encoding="UTF-8"?>
<poolman>
    <management-mode>local</management-mode>
    <datasource>

        <dbname>jonas_db1</dbname>
        <jndiName>jdbc_1</jndiName>
        <driver>com.mysql.jdbc.Driver</driver>
        <url>jdbc:mysql://www-tp.int-evry.fr/jonas_db1</url>

        <username>jonas_db1_user</username>
        <password>ujdb1</password>
        <minimumSize>0</minimumSize>
        <maximumSize>5</maximumSize>
    </datasource>
</poolman>
```

<maximumSize> désigne le nombre de connexions que le regroupement peut contenir. Ici par exemple, chaque regroupement contient 5 connexions.

-Placer ce fichier dans le répertoire $JONAS_BASE/conf

-Dans les fichiers des EJB (ici BiddingBean, UserBean et ProductBean) remplacer l'ancienne méthode « getConnection() » par celle-ci :

```
      private Connection getConnection() throws EJBException, SQLException
{
         try {

            Class.forName("com.codestudio.sql.PoolMan").newInstance();

            } catch (Exception ex) {
                              System.out.println("Could    Not    Find    the
YAPoolMan Driver. \n" +
                              "Is poolman.jar in your CLASSPATH?");
                      }

                      Connection con = null;
                      try {

                              //  establish  a  Connection  to  the  database
with
                              // <dbname>testdb</dbname>
                              //in the poolman.xml file
                        con = DriverManager.getConnection("jdbc:poolman");

                      } catch (SQLException sqle) { sqle.printStackTrace();
}

                      return con;

              }
```

5.3. Les tests réalisés

Lien	**Sans poolMan**	**Avec poolMan**
Poser une enchère sur un produit : http://elaphe.int-evry.fr:9000/SiteEnchere/bidTest.jsp?price=…	-2 clients : 130ms -5 clients : 158ms -15 clients 176ms -16 clients (ça plante).	-2 clients : 132 ms. -15 clients : 160ms. -16 clients ça plante.
Connexion au site : http://elaphe.int-evry.fr:9000/SiteEnchere	On peut atteindre 1000 clients simultanés avec 380 ms.	Pareil pour 1000 clients 350 ms.
Authentification : Même pseudo=oussama http://elaphe.int-evry.fr:9000/SiteEnchere/secured/authentication.jsp?user=oussama	-100 clients : 2407 ms -300 clients ça plante.	-100 clients 2000ms. -250 ça plante.
Liste des produits disponibles http://elaphe.int-evry.fr:9000/secured/BiddingOnProductTest.jsp	-1 client 511ms -2 clients 830ms -5 clients 2171ms -10 clients 2516 ms.	-1 client: 385 ms -2 clients : 711ms -3 : 1100ms -5 clients : ça plante.
Autres enchères faites par un utilisateur http://elaphe.int-evry.fr:9000/SiteEnchere/secured/autresEncheresTest.jsp?pseudo=oussama	-1 : 444 ms -2 clients 673 ms. -3 clients ça plante	-1 client : 715ms -2 clients : 711ms -3 clients : 952ms -4 clients ça plante.
Autres enchères sur un produit : http://elaphe.int-evry.fr:9000/SiteEnchere/secured/biddingsOnAProductTest.jsp	-1 client : 274ms -3 clients : 505ms -6 clients ça plante.	-1 client : 250 ms -3 clients 500 ms -5 clients ça plante.

On constate, d'après ces mesures, que l'outil poolMan n'a pas vraiment contribué à l'amélioration des performances de l'application vu que les temps de réponse du serveur dans les deux cas (avec et sans poolMan) sont très proches et vu que ce logiciel n'a pas pu résoudre les problèmes de blocage.

Dans les deux cas, on constate que plus le nombre de clients qui accèdent simultanément à un lien augmente, plus le temps de réponse augmente mais cette augmentation n'est pas linéaire et se fait en plus de façon aléatoire (lorsqu'on effectue plusieurs expériences sur le même lien on constate que le temps que met le serveur pour répondre avec un même nombre de clients varie) car ceci est lié à l'état du réseau et à d'autres facteurs externes.

Nous n'avons pas pu, malheureusement, résoudre les problèmes de blocage mais nous pensons qu'ils sont liés à la capacité mémoire de la machine « elaphe » qui est partagé par « tous » les étudiants de l'INT et qui en plus, dans le cadre de notre projet, sert de serveur d'application et héberge la base de données.

II- Utilisation des composants OpenCCM

Cette partie est consacrée à l'utilisation d'openCCM ainsi qu'à la réalisation de notre site d'enchère grâce aux composants Corba. Nous allons introduire cette partie avec le mode d'installation des différents éléments indispensables à l'utilisation d'OpenCCM. Puis seront explicité les diverses aspects d'openCCM à savoir la définition des CCM, l'explication du modèle abstrait ainsi que l'explication du mode d'utilisation des composants. Nous verrons ensuite la transcription de notre application de site d'enchère à travers les composants CCM. Cette partie sera suivie d'un « mode d'emploi » expliquant la marche que nous avons suivi lors de la création de notre application. Nous finirons cette partie par les problèmes rencontrés et les tests utilisés.

1. Presention d'OpenCCM

1.1 L'installation

Cette partie peut prendre beaucoup de temps et il est important de ne pas la négliger.

- **Installation du JDK 1.4.2**

Pour l'utilisation d'openCCM il est nécessaire d'avoir un Java Developpement Kit (JDK)
- Téléchargez j2sdk-1_4_2 (binaire) sur le site http://java.sun.com/j2se/1.4.2/download.html
- Modifiez les droits du fichier binaire en tapant la commande :
    $chmod +x j2sdk-1_4_2<version>-linux-i586.bin
- Exécutez le fichier binaire en tapant la commande :
    $./ j2sdk-1_4_2<version>-linux-i586.bin
- Modifiez la variable JAVA_HOME dans le .bashrc
    Export JAVA_HOME = <chemin ou se trouve java>

- **Installation de l'ORB**

Pour notre projet nous avons utilisé **OpenORB** (mais on aurait pu également utiliser JacORB ou ORBacus).
- Téléchargez sur le site http://sourceforge.net/project/showfiles.php?group_id=43608
  OpenORB-1.4.0.tgz NameService-1.4.0.tgz SSL-1.4.0.tgz Tools-1.4.0.tgz
- Extraire les archives avec la commande :
    $tar xvzf <nomDeLArchive>.tgz
- Modifiez le fichier .bashrc en incluant les lignes :
    export OpenORB_HOME=chemin où se trouve l'ORB
    export CLASSPATH=$CLASSPATH :.:$OpenORB_HOME/lib
- Création du fichier orb.properties en tapant la commande :
    $JAVA_HOME/bin/java –jar $OpenORB_HOME/lib/open_orb-1.4.1.jar

- **Installation de l'OpenCCM**

- Téléchargez sur le site http://sourceforge.net/project/showfiles.php?group_id=10
    OpenCCM-0.8.3.tar.gz
- Extraire l'archives avec la commande
    $tar xvzf OpenCCM-0.8.3.tar.gz
- Allez dans le répertoire OpenCCM-0.8.3/openccm
- Exécuter la commande :
    $build.sh
    Un message d'erreur s'affiche mais le fichier build.properties s'est crée.
- Ouvrir le fichier build.properties et modifier les champs suivants :

```
# ========================================================================
#
# Select the ORB used to compile and execute the OpenCCM platform.
#
# Uncomment the used ORB and comment other ORBs.
#
# ========================================================================

# For OpenORB-1.4.0 for Java.
#
ORB.name=OpenORB-1.4.0
# ========================================================================
#
# The directory where the used ORB is installed.
#
# Must be set!
#
# Warning: On Windows systems, use / instead of \ as directory separator.
ORB.home.dir=chemin ou se trouve l'ORB
# ========================================================================
```

- Recompilez en exécutant la commande
  $build.sh (la compilation dure environ une dizaine de minutes).
- Afin d'installer OpenCCM on exécute la commande suivante :
  $build.sh install

OpenCCM est prêt à l'emploi…

1.2 Aspect d'OpenCCM

Le CCM est un *framework* de composants de type serveurs, mais qui peut être utilisé coté client. Cette technologie repose sur l'utilisation de conteneurs qui servent d'environnement d'exécution des instances composants, ces conteneurs offres des services à ces instances

- Le modèle abstrait : il offre la possibilité de définir des interfaces et les propriétés d'un type de composant, il permet aussi de définir les gestionnaires d'instances de composants. Le langage utilisé est l'OMG IDL.

- Le modèle de programmation : il spécifie le langage CIDL (Component Implémentation définition Language) à utiliser pour définir la structure de l'implantation d'un type de composant, ainsi que certains de ces aspects non-fonctionnels (persistance, transactions, sécurité). L'utilisation de ce langage est associé à un *framework,* Le CIF (Component Implementation Framework) qui définit comment les parties fonctionnelles (programmées) coopère avec les parties non-fonctionnelles(décrites en IDL/CIDL est générées). Il définit aussi la manière dont le composant interagit avec le container.

- Le modèle de déploiement : il sert à installer une application sur différents sites d'exécution de manière simple et automatique.

- Le modèle d'exécution définit l'environnement d'exécution des instances de composants. Le rôle principal des containers est de masquer et prendre en charge les

aspects non-fonctionnelelles les composants qu'il n'est alors plus nécessaire de programmer.

- **Le modèle abstrait OpenCCM**

Le modèle abstrait va nous permettre de définir les diverses interfaces fonctionnelles nécessaires à la communication entre les différents composants. Ce modèle utilise le langage idl3 afin de définir ces interfaces. Puis lors de la phase de compilation un fichier idl2 est généré afin de permettre au développeur d'engendrer le stub côté client et le skeleton côté serveur.
Ceci est décrit dans les schémas suivants disponibles dans le guide d'utilisateur sur le site d'openCCM.

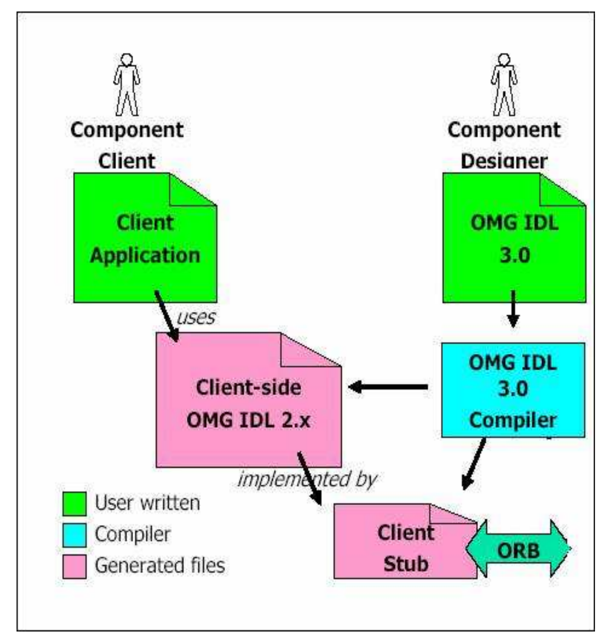


Figure 1 : la chaîne de compilation de composant CORBA de point de vue client

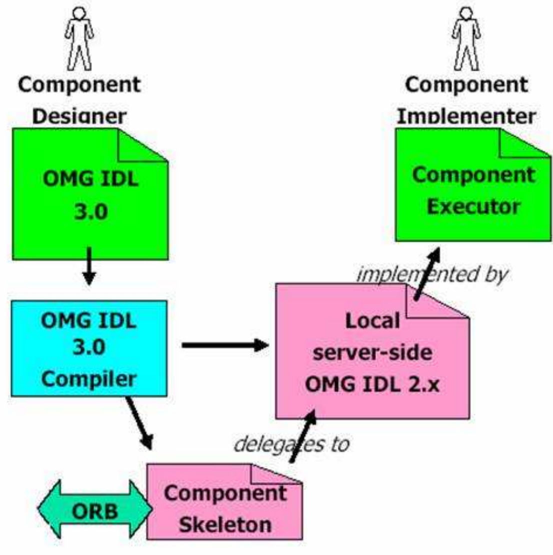
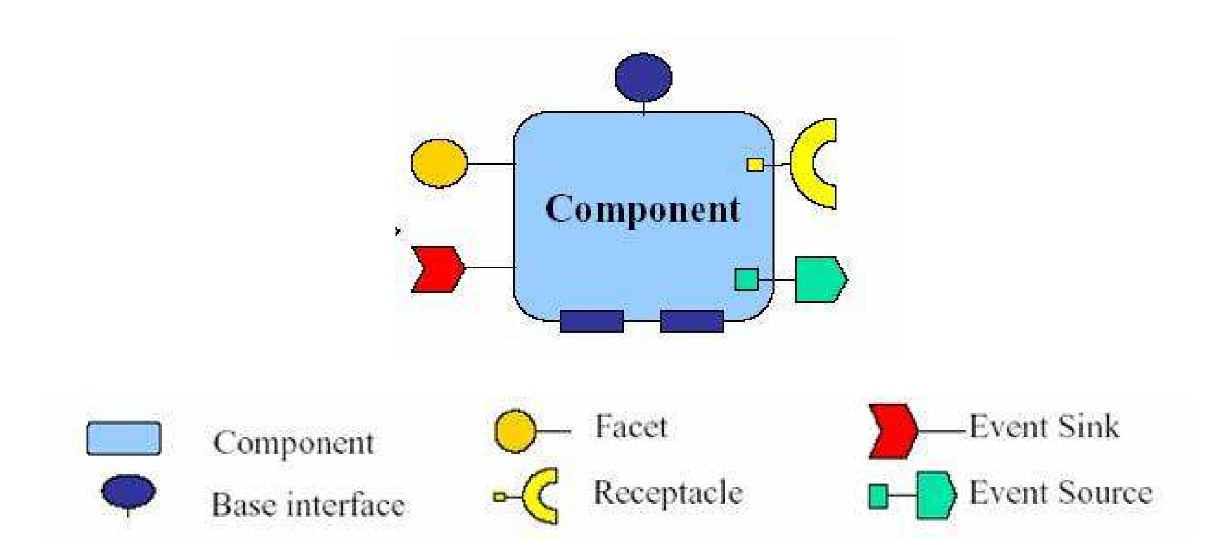


Figure 2 : la chaîne de compilation de composant CORBA de point de vue serveur

Ces schémas symétriques montre bien l'unicité du fichier IDL3 qui va fournir la description du mode d'interaction entre le client et le serveur. Nous reviendrons dans la partie sur la marche à suivre pour créer nos différents composants et les mettre en relation.

- **Définition d'un composant**

Dans le guide utilisateur un composant est défini de la manière suivante :



Chaque composant doit implémenter une interface prédéfinie qui va permettre de définir les propriétés du composant ainsi que l'ensemble des opérations de contrôle.
Il hérite de l'interface **Components :: CCMObject.**

Chaque composant à la possibilité de communiquer avec d'autres composants. Pour cela ils ont à leurs dispositions 2 paires d'outils de communications :
Réceptacle / Facette et Event Sink / Event Source.

Event Sink / Event Source
Nous n'avons pas utilisé ce mode lors de notre projet. Les events source vont permettre de générer des évènements et les events sink de les consommer.

Réceptacle / Facette
La facette va mettre a disposition une référence d'objet et le recpetacle va permettre au composant d'accepter cette référence. Ce jeu de réceptacle/facette doit être définit explicitement dans le fichier .idl3 car il va retranscrire une interface.

2. Notre application sous OpenCCM

Nous allons exposer dans cette partie la conception de notre application. Nous introduisons tout d'abord les composants utilisés en utilisant la représentation standard et en expliquant le rôle de chacun d'entre eux dans l'application. Puis un schéma résumant l'interaction entre ses composants sera présenté. Il sera suivi de la caractérisation des interfaces introduites dans le schéma de relation des composants. Le schéma relationnelle de base de données ainsi le schéma représentant l'enchainement des interfaces graphiques sont les mêmes que dans la première partie.

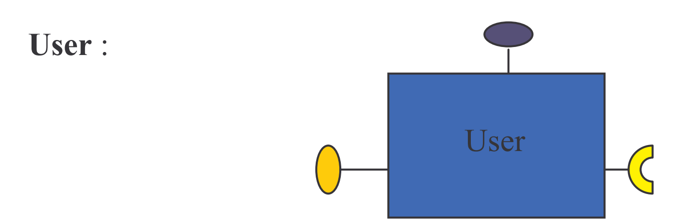
2.1 Les composants

Client GIU:



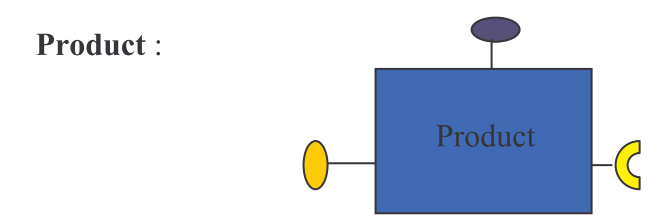
Composant qui va servir de relais entre le client et le traitement. Dans ce composant l'on retrouve les différents formulaires utiles pour les différentes opérations d'enchère/vente.

User :



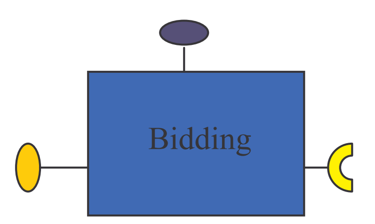
Composant tampon entre le client et la base de données. Il va permettre d'avoir une vue locale des informations du client. Entre la demande de création et la validation de cette création du client au niveau du serveur, ce composant va servir à stocker au niveau local des informations du client.

Product :



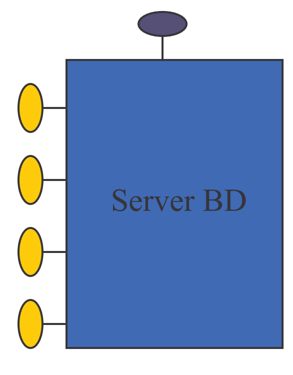
Composant tampon entre le produit et la base de données. Comme le composant User il va permettre d'avoir une vue locale des informations du produit et donc de stocker également au niveau local des données du produit.

Bidding :



Composant de transition des applications d'achat/vente entre le client et le serveur de base de données.

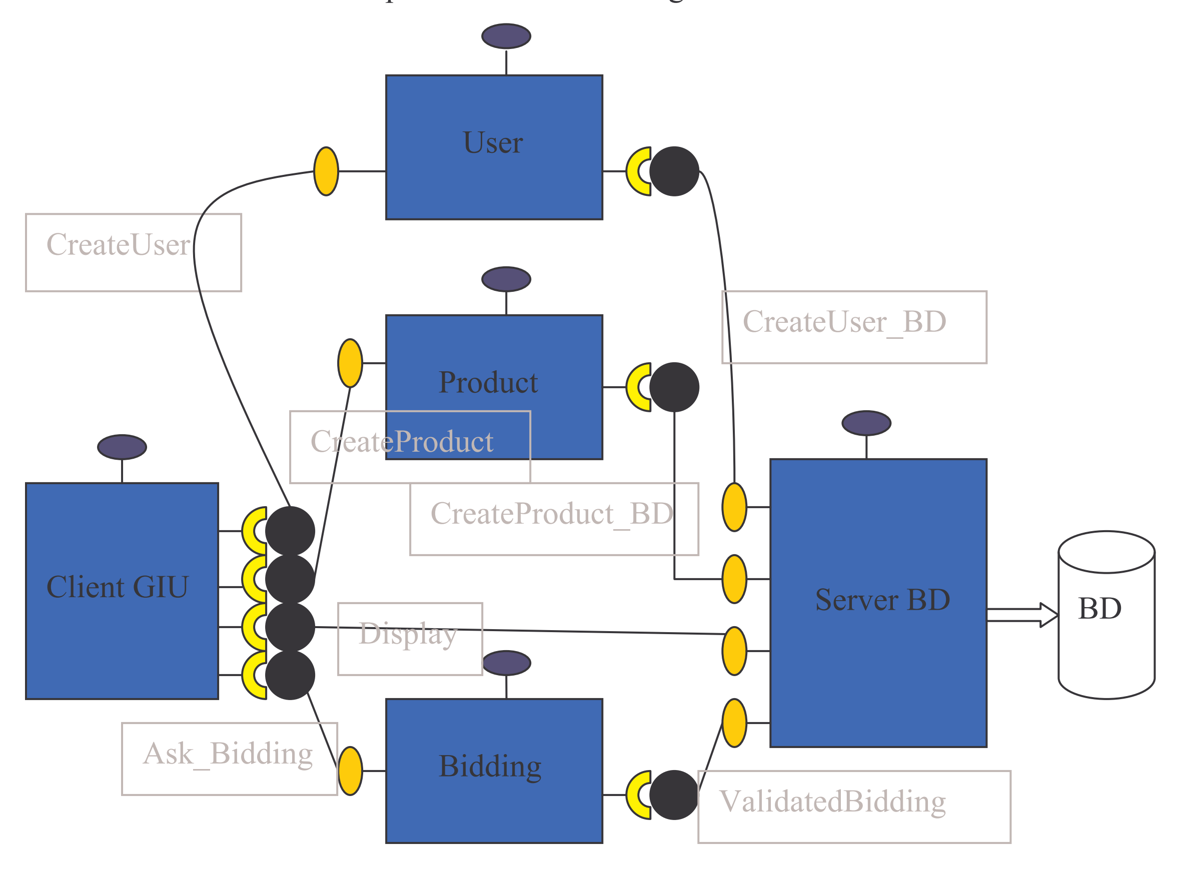
Server BD :



Composant qui va gérer le stockage des informations ainsi que de l'envoie de la confirmation de vente et d'achat.

2.2 Le schéma de communication de nos composants

Les interfaces entre deux composant sont écrites en gris clair



2.3 Les interfaces

AskBidding: va permettre la demande de vente/enchère.

// procedure de demande de vente avec les paramètres
```
void sell(in string pseudo_seller, in long id_product,in float start_price,in long step,in string date)
//fonction de demande d'enchère.Si la date d'enchère est dépassée elle renvoie faux
boolean bidding(in string pseudo, in float new_price,in long id_product);
```

Validated_Bidding: va confirmer la transaction de vente/enchère au niveau du stockage.

```
/*fonction de validation d'enchère. Elle va donc faire la junction des diverses tables de la base de
données et stocker les informations. Si la date est expirée elle renvoie faux*/
boolean bidding_validated(in string pseudo, in float new_price, in long id_bidding);
/*fonction de validation de vente. Elle va donc faire la junction des diverses tables de la base de
données et stocker les informations*/
void sell(in string pseudo_seller, in long id_product,in float start_price,in long step,in string date);
//function qui renvoie le numero d'identité de l'enchère
long getIdBidding();
```

Create_User: va contenir une fonction demandant la création d'un utilisateur ainsi qu'une fonction de récupération des informations sur ce même utilisateur. Cette interface va permettre également une demande d'authentification du client.

//fonction de demande de creation d'utilisateur. Si le pseudo existe déjà elle retourne faux
```
boolean create_a_user(in string pseudo,in string nom,in string adress,in string mail);
//fonction qui retourne le pseudo de l'utilisateur
string getPseudo();
// function demandant l'authentification du client
boolean authentify(in string login,in string pw);
```

***Create_Product* :** de même que pour Create_User elle va engendrer la demande de création d'un objet produit et va permettre la récupération de certaines données.

//fonction de demande de creation d'un produit
```
void create_a_product(in string title,in string categorie,in string description);
//function de recuperation du numero d'identification du produit
long getIdProduct();
```

Create_User_BD: est composée d'une fonction de validation de la création de l'utilisateur au niveau de la base de données ains qu'une fonction de validation d'authentification.

/*fonction de validation de creation de l'utilisateur. Si l'utilisateur existe dejà elle retourne faux*/
```
boolean  create_a_user_BD(in string pseudo,in string nom,in string adress,in string mail);
//function de validation d'authentification
boolean authentify(in string login,in string pw);
```

***Create_Product_BD* :** de même que pour Create_User_BD, elle va permettre la validation de la création du produit au niveau de la base de données.

```
//fonction de validation de la creation d'un produit
void create_a_product_BD(in string tilte,in string categorie,in string description);
//fonction de recuperation du numero d'identification du produit
long getIdProduct();
```

Display: Elle va également permettre le listage au niveau des enchères stockées dans la base de données.

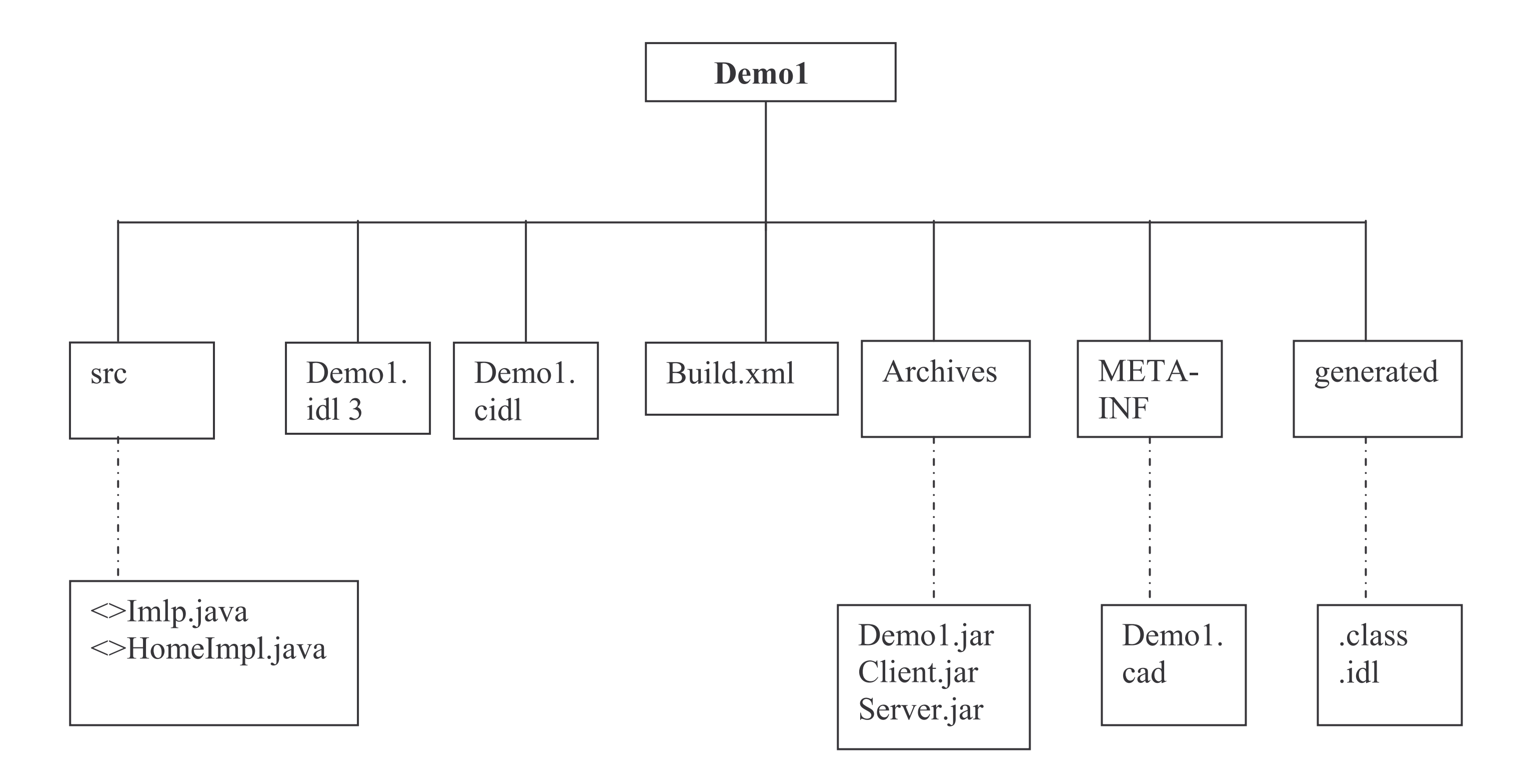
```
Object_data_list listBidding();
```

Pour pouvoir communiquer les informations de la base de données au client nous avions besoin de déclarer une structure **Object_data** qui va contenir tous les attributs de la base de données concernant l'enchère. Pour pouvoir lister les enchères nous avons utilisé un tableau d'Object_data nommé **Object_data_list**.

}

3. La marche à suivre

Afin de mieux comprendre le fonctionnement d'openCCM nous avons suivi l'exemple de base la demo1. Nous nous sommes donc servi de l'existent afin de gagner du temps (disposition d'un moyen direct de compilation et d'éviter de toucher au build.xml et au build.sh.). Afin de pouvoir situé les fichiers que nous avons modifiés nous avons représenté l'architecture du répertoire demo1.



Nous allons décrire les différentes étapes à suivre en incluant des exemples.

La première étape est d'écrire dans le fichier .idl3. Ce fichier contient la définition des composants, des structures ainsi que des interfaces.

- **Exemple de déclaration d'interface avec Create_User**

```
interface Create_User
{

        boolean create_a_user(in string pseudo,in string nom,in string adress,in string mail);
        string getPseudo();
        boolean authentify(in string login,in string pw);
};
```

- **Exemple de déclaration de structure avec Object_data et Object_data_list**

```
struct Object_data
        {
        string title;
        string category;
        string description;
        float start_price;
        float step;
        boolean status;
        float actual_price;
        long id;
        string date_end;
        };

typedef sequence<Object_data> Object_data_list;
```

Attention lorsque la structure est utilisée dans un fichier il faut lui ajouter à un import pour que lors de la compilation reconnaisse la structure.

- **Exemple de déclaration de composant avec Product**

```
   component Product
  {
     /** The facet for Clients components. */
    provides Create_Product for_clients_from_Product;

    /**
      * The receptable to_bidding to connect the Client component
      * to a validation_Bidding object or facet reference.
      */
     uses Product_BD to_server_from_product;
  };
     home ProductHome manages Product
   { };
```

La deuxième étape est d'écrire dans le fichier .cidl permettant la déclaration explicite des composants. Pour cela on utilise CIF (Component Implementation Framework) qui va décrire les interactions entre la partie fonctionnelle et non fonctionnelle de chaque composant

- **Exemple de déclaration du composant Client**

```
  composition session ClientSessionComposition
  {
     home executor HomeImpl
     {
        implements ClientHome;
        manages ComponentImpl;
     };
  };

  composition session ServerSessionComposition
  {
     home executor HomeImpl
     {
```

```
        implements ServerHome;
        manages ComponentImpl;
    };
};
```

La troisième étape est l'écriture des fichiers <nomComposant>HomeImpl.java et <nomComposant>Impl.java avec le code souhaité.
Dans <nomComposant>Impl.java on implémente les méthodes offertes par le composant (défini dans l'IDL) ainsi qu'une méthode configure qui sera être appelée au moment du lancement de l'application.

<nomComposant>HomeImpl.java qui va permettre de créer et d'instancier des composants.

- **Exemple avec BiddingHomeImpl.java**

```
public class BiddingHomeImpl
     extends org.objectweb.ccm.demo1.BiddingSessionComposition.HomeImpl
{

    /** The default constructor. */
    public
    BiddingHomeImpl()
    {
    }
    /**
     * Create an executor segment from its identifier.
     *
     * @param segid The executor segment identifier.
     */
    public org.omg.Components.ExecutorSegmentBase
    create_executor_segment(int segid)
    {
        return new BiddingImpl();
    }

    // ====================================================================
    //
    // Methods for the deployment.
    //
    // ====================================================================

    /**
     * This method is called by the OpenCCM Component Server
     * to create a home instance.
     */
   public static org.omg.Components.HomeExecutorBase
    create_home()
    {
        return new BiddingHomeImpl();
    }
}
```

La quatrième étape est la jonction des différents composants, cela se passe dans le fichier demo1.java où l'on trouve le main

- **Exemple avec l'introduction du composant User :**

```
//composant serveur
org.objectweb.ccm.Deployment.Server server5 =
org.objectweb.ccm.Deployment.ServerHelper.narrow(obj);
…………………………………………………………………………
//obtention des container homes et des installateurs d'archives
  org.omg.Components.Deployment.ComponentServer server5_cs =
server5.provide_component_server();
         org.omg.Components.Deployment.ComponentInstallation server5_inst =
server5.provide_install();
……………………………………………………………………………
server5_inst.install("demo1", "file:"+demoPath + "./archives/demo1.jar");
……………………………………………………………………………..
server5_cs.create_container(new org.omg.Components.ConfigValue[0]);
…………………………………………………………………………….
org.omg.Components.CCMHome h ;//dejà existant
……………………………………………………………………………..
  h = server5_cont.install_home("demo1",
                              "org.objectweb.ccm.demo1.cif.UserHomeImpl.create_home",
                               new org.omg.Components.ConfigValue[0]);
            UserHome uh= UserHomeHelper.narrow(h);
……………………………………………………………………………
         Server s = sh.create();//deja existant
         Client c1 = ch.create();//deja existant
          User u= uh.create();
……………………………………………………………………………
//utilisation des interfaces
Create_User for_clients_from_user=u.provide_for_clients_from_user();
User_BD for_User_from_server=s.provide_for_user_from_server();
//en rouge on été declare lors de la declaration du composant de le fichier idl3
//avec l'exemple de  Product on aurait utilisé for_clients_from_Product
……………………………………………………………………………
//connect le client a l'utilisateur et l'utilisateur au serveur
c1.connect_to_user(for_clients_from_user);
u.connect_to_server_from_user(for_User_from_server);
……………………………………………………………………………
// Configuration completion.
u.configuration_complete();
```

La cinquième étape est de modifié le fichier .cad dans le repertoire META-INF pour pouvoir déployer les composants.

- **Exemple avec l'introduction de Bidding**

```
</homeplacement>
   <homeplacement cardinality="1" id="BiddingHome">
     <componentfileref idref="BiddingCSD"/>
     <componentimplref idref="BiddingImpl"/>
     <registerwithhomefinder name="demo1-BiddingHome"/>
     <registerwithnaming name="demo/demo1/BiddingHome"/>
     <registerwithtrader>
       <traderexport>
         <traderservicetypename>HomeService</traderservicetypename>
         <traderproperties>
           <traderproperty>
```

```
          <traderpropertyname>description</traderpropertyname>
          <traderpropertyvalue>demo1: The Bidding home</traderpropertyvalue>
        </traderproperty>
      </traderproperties>
    </traderexport>
  </registerwithtrader>
  <componentinstantiation id="Mathieu">
    <componentproperties>
      <fileinarchive name="META-INF/mathieu.cpf">
      </fileinarchive>
    </componentproperties>
    <registercomponent>
      <registerwithtrader>
        <traderexport>
          <traderservicetypename>ComponentService</traderservicetypename>
          <traderproperties>
            <traderproperty>
              <traderpropertyname>description</traderpropertyname>
            </traderproperty>
          </traderproperties>
        </traderexport>
      </registerwithtrader>
    </registercomponent>
  </componentinstantiation>
    <registercomponent>
      <registerwithtrader>
        <traderexport>
          <traderservicetypename>ComponentService</traderservicetypename>
          <traderproperties>
            <traderproperty>
              <traderpropertyname>description</traderpropertyname>
              <traderpropertyvalue>demo1: The Philippe component</traderpropertyvalue>
            </traderproperty>
          </traderproperties>
        </traderexport>
      </registerwithtrader>
    </registercomponent>
  </componentinstantiation>
        <traderexport>
          <traderservicetypename>ComponentService</traderservicetypename>
          <traderproperties>
            <traderproperty>
              <traderpropertyname>description</traderpropertyname>
            </traderproperty>
          </traderproperties>
        </traderexport>
      </registerwithtrader>
    </registercomponent>
  </componentinstantiation>
```


....


```
   <connectinterface id="server-Bidding">
  <usesport>
    <usesidentifier>to_Bidding</usesidentifier>
    <componentinstantiationref />
  </usesport>
  <providesport>
    <providesidentifier>for_clients_from_Bidding</providesidentifier>
    <componentinstantiationref idref="Bidding"/>
  </providesport>
 </connectinterface>
```

A ce stade l'on peut compiler en lançant : build.sh
Une fois la compilation réussie on exécute le programme en tapant : bin/start_java

4. Les problèmes

Nous avons essayé d'appliquer les pages jsp utilisé dans la partie J2EE mais nous avions eu un problème de déploiement au niveau de tomcat. En effet nous avions essayé de joindre la page jsp et notre application de traitement à travers un bean en mettant ainsi les les fichiers .class (obtenu a la fin de la compilation se situant dans le repertoire archives dans l'archive client.jar) sous le repertoire WEB-INF/classes de tomcat. Tomcat ne reconnaît pas les indispensable au bon fonctionnement. Nous avons donc rajouté à notre CLASSPATH ces archives mais le problème fut persistant.

Nous avons également essayé d'utiliser le caractère de persistance d'openCCM. Mais la documentation sur ce sujet est assez restrictive, et elle nécessite une clé de licence de JDO.

L'installation de OpenCCM a échoué sur nos comptes mci nous n'avons pou tester comme dans la première partie la charge de clientèle au niveau de la base de données. Nous nous sommes limité à faire des tests métiers (comme tester le time out).

III-Comparaison des avantages et inconvénients des deux modèles

Points de comparaison	**OpenCCM**	**EJB**
Installation	- Installation un peu difficile mais possibilité de choix d'ORB. - Il faut faire attention à l'environnement dans lequel on va installer le serveur.	- Demande beaucoup d'espace disque. - Installation un peu difficile. - Il faut faire attention à l'environnement dans lequel on va installer Jonas (version de JDK, version de Tomcat…).
Déploiement	- Pas de documentation : il faut juste se baser sur des exemples basiques.	- Pas de documentation : il faut juste se baser sur des exemples. - Il faut faire attention à l'arborescence des fichiers. - Points positifs : -- Utilisation de Ant pour la compilation. -- Interface graphique fournies par le serveur Jonas pour la mise en œuvre de l'application.
Compatibilité avec l'environnement Web	- Pas de compatibilité connue. - Pas d'exemples de persistance.	- Compatible avec l'environnement J2EE : JSP/Servlet, JavaBeans… - Accès à la base de données assez simple.
Documentation	- Légère. - Il faut se baser surtout sur les exemples pour le développement.	- La documentation se trouve principalement sur le site d'Object Web. Son défaut est qu'elle est ample mais non ciblée=> ne convient pour un débutant. - Il faut se baser surtout sur les exemples pour le développement.

A l'issue de ce projet, et en comparant les deux versions, nous pensons que l'utilisation des EJB pour développer une application est plus adaptée. De plus, nous avons constaté que OpenCCM est plutôt utilisé pour des projets de recherche.

Conclusion

Nous avons présenté dans ce rapport les démarches suivies pour implémenter notre site d'enchère avec les techniques EJB et openCCM, les différentes difficultés que nous avons rencontrées et une comparaison entre ces deux technologies.

Ce projet fut une occasion de découvrir des outils open source et d'évaluer leurs performances même si ce fut dans le cadre d'une application simple mais très enrichissante pour des débutants.

Ce projet correspond à la continuité de l'enseignement reçu dans l'option ASR : en effet à travers ce projet nous avons utilisé des technologies apprises durant notre cursus (J2EE et CORBA) tout en augmentant nos compétences grâce à l'apprentissage des mécanismes d'OpenCCM et d'EJB.